Netconf 2023

Eric Dumazet @ Google

Agenda

01 Struct file reorganization for networking

02 dev_queue_xmit_nit() made better

03 Defer wakeups in TCP/SCTP/MPTCP...

04 FQ with 3 bands and WRR

05 UDP listen()/accept()/and 4-tuple lookups

Struct file is silly for networking

```
commit a7bc2e8ddf3c8e1f5    fs.h: Optimize file struct to prevent false sharing


After this commit, we use 4 cache lines from 'struct file' per socket system call.

fmode_t                     f_mode;             /*  0x14   0x4 */
atomic_long_t               f_count;            /*  0x18   0x8 */

unsigned int                f_flags;            /*  0x48   0x4 */

const struct file_operations  * f_op;           /*  0xb0   0x8 */

void *                      private_data;       /*  0xc8   0x8 */

Cited commit claims an unixbench improvement...
```

Struct file better layout ?

Keep heavily modified field in separate cache line (to not break benchmarks ?)

But place f_op, private_data, f_flags and f_mode in a single cacheline.

Do not clone packets in dev_queue_xmit_nit()

Maciej Żenczykowski raised the issue on netdev@ in July (Performance question: af_packet with bpf filter vs TX path skb_clone)

Cloning packets in order to run BPF filters and drop 99% of them makes little sense.

It would be better to run BPF filter, then clone the skb if we expect to queue it in **af_packet**.

We do not need to add a parameter to ptype->func(), we might simply use skb->pkt_type == PACKET_OUTGOING to determine if cloning is needed after run_filter() ?

Also need to defer net_timestamp_set(skb) to not mess with EDT.

Defer wakeups in TCP/SCTP/MPTCP/…

sk->sk_data_ready() and friends are usually called while holding the socket lock.

This leads to high spinlock contention in some situations

epoll() and/or process scheduler logic can be quite expensive (cache line bouncing).

Idea would be to postpone these calls after socket lock is released (under rcu read lock).

FQ with 3 prios and DRR

Google uses a specialized version of FQ, adding 3 prios/bands (like **pfifo_fast**) and WRR scheduling (unlike strict prio of **pfifo_fast** or **prio**)

This replicates some of DRR or PRIO functionalities, but with less hops and better scheduling after a watchdog timer completion.

Time to upstream, after my recent addition of a fast path in FQ ?

UDP listen()/accept()/4-tuple lookups

Might help some QUICK servers ?

Answer: Use BPF with SO_REUSEPORT instead ?